Food Critics Sentiment Analysis using Neural Network

Presented By: Utsaw Siwakoti

ID: 6019449

Subject: Artificial Intelligence

Subject Code: SC6360

Presented To: Asst. Prof. Dr. Anilkumar K. G

Contents

- Introduction
- Current Scenario and Objectives
- Dataset and Architecture of the proposed System
- Proposed System
- Glance at the System
- Conclusion

Introduction

- The system is about analyzing the sentiment of food critics of a restaurant using Deep Learning with Neural Network.
- The system uses TensorFlow library by Google for machine learning.

Current Scenario and Objectives

- There are various people who give comments on the food and service provided by a restaurant on their website. It would be very beneficial for a restaurant to analyze these comments and enhance their food service.
- But, it is very difficult to analyze these numerous sentiments in person. Hence, the use of sentiment analysis using Neural Networks.
- The objectives of the proposed system are:
  - To identify the positive and negative feedback given by a customer on their website and analyze them for future reference.
  - Training the pre-existed dataset for Machine Learning.
  - Analyzing test data with trained classifier using TensorFlow.
  - Predicting a comment being positive or negative.
  - Providing accuracy analysis.

Dataset and Architecture of the proposed system

- The Dataset used in this system is provided by University of California, Irvine Department of Computer Science.
- The dataset has 400 positive 400 negative food critics by their customers on their website.
- The system is designed using Python Programming Language and uses TensorFlow library for Deep Learning with Neural Network.
- There are various keywords in the sentences which needs to be processed. Natural Language Processing is done by using Rapid Automatic Keyword Extraction (RAKE) algorithm in Python Programming.

Proposed System

- At first the system loads the train dataset, test dataset and uses RAKE algorithm to grab unique words to train for Machine Learning.
- The Network is then generated using tflearn library and then saved as a classifier model.
- After that, a test dataset is provided which is confronted against the saved “classifier model”to finally execute the prediction of the sentiment.
- The Evaluation Result, precision, recall and accuracy are executed as output of the system.

Glance at the System

- Training the Dataset

```
PS C:\Users\Utsaw> cd AI
PS C:\Users\Utsaw\AI> .\venv\Scripts\activate.ps1
(venv) PS C:\Users\Utsaw\AI> python fc.py
hdf5 is not supported on this machine (please install/reinstall h5py for optimal experience)
curses is not supported on this machine (please install/reinstall curses for an optimal experience)
Scipy not supported!
---------------------------------
Run id: fc-classifier
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 800
Validation samples: 200
--
Training Step: 1  | time: 1.000s
| Adam | epoch: 001 | loss: 0.00000 - acc: 0.0000 -- iter: 064/800
←[A←[ATraining Step: 2  | total loss: ←[1m←[32m0.62383←[0m←[0m | time: 1.000s
| Adam | epoch: 001 | loss: 0.62383 - acc: 0.4781 -- iter: 128/800
←[A←[ATraining Step: 3  | total loss: ←[1m←[32m0.68084←[0m←[0m | time: 1.016s
| Adam | epoch: 001 | loss: 0.68084 - acc: 0.4832 -- iter: 192/800
```

- Evaluate Test Data

```
(venv) PS C:\Users\Utsaw\AI> python fc.py
hdf5 is not supported on this machine (please install/reinstall h5py for optimal experience)
curses is not supported on this machine (please install/reinstall curses for an optimal experience)
Scipy not supported!
Evaluation result: [0.80500000000000005]
probability of comment being negative: 0.5983052849769592
probability of comment being positive: 0.4917358160018921
probability of comment being negative: 0.5983052849769592
precision= 0.5625
recall= 0.6
accuracy= 0.805
```

Conclusion

- The evaluation result of the system is around 80%, which means the results are pretty accurate.
- Using Deep Learning with Neural Network to train and analyze datasets can be very helpful to enhance business strategies and marketing techniques.

Appendix

```
"""
Food Critics training with deep learning
Author: Utsaw Siwakoti, 2017
"""

from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import base64
import numpy as np

import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
from rake_nltk import Rake
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# clean dataset
def clean_data():
    f = open('fc.txt', 'r')
    sentence = ''
    for line in f:
        # removing , . ! from the sentence
        data = line.strip().replace(',','').replace('.','').replace('!','').split(';;')
        sentence += data[1]+','+data[0].replace(';', '').lower()+'\n'
    f = open('fc-clean.csv', 'w')
    f.write(sentence)

# grab uniquewords
def get_uniquewords(file_path):
    uniquewords = []
    f = open(file_path, 'r')
    for line in f:
        text = line.strip().split(',')[1]
        if text != 'text':
            words = text.split(' ')
            for word in words:
                if word not in uniquewords:
                    uniquewords.append(word)
    return uniquewords

# preprocess for data
def preprocess(file_path, unique_words):
    sentences = []
    vectors = []
    f = open(file_path, 'r')
    for line in f:
        text = line.strip().split(',')[1]
        if text != 'text':
            words = text.split(' ')
            inner_data = []
            for word in words:
                inner_data.append(word)
            sentences.append(inner_data)
    for sentence in sentences:
        inner_vector = []
        for word in unique_words:
            if word in sentence:
                inner_vector.append(1)
            else:
                inner_vector.append(0)
        vectors.append(inner_vector)

    return np.array(vectors, dtype=np.float32)


train_file_path = 'fc-clean.csv'
test_file_path = 'fc-clean-test.csv'

from tflearn.data_utils import load_csv
data, labels = load_csv(train_file_path, target_column=0, categorical_labels=True, n_classes=2)
testdata, testlabels = load_csv(test_file_path, target_column=0, categorical_labels=True, n_classes=2)

f = open('fc-clean.csv', 'r')
sentences = ''
for line in f:
    text = line.strip().split(',')[1]
    sentences = sentences+text+' '

r = Rake(language='english')
r.extract_keywords_from_text(sentences)
uniquewords = r.get_ranked_phrases()

words = uniquewords

uniquewords = []
for word in words:
    each_word = word.split(' ')
    for w in each_word:
        uniquewords.append(w)

# grabbing first 200 unique words for vector
uniquewords = uniquewords[:200]
```

```
# uniquewords = get_uniquewords(train_file_path)
data = preprocess(train_file_path, uniquewords)
testdata = preprocess(test_file_path, uniquewords)

neurons = len(data[0])

# shuffle the dataset
from tflearn.data_utils import shuffle
data, labels = shuffle(data, labels)

network = input_data(shape=[None, neurons])
network = fully_connected(network, 32, activation='relu')
network = fully_connected(network, 32*2, activation='relu')
network = fully_connected(network, 32, activation='relu')
network = dropout(network, 0.5)

network = fully_connected(network, 2, activation='softmax')
network = regression(network, optimizer='adam', learning_rate=0.01, loss='categorical_crossentropy')

model = tflearn.DNN(network)
# uncomment this for training
#model.fit(data, labels, n_epoch=20, shuffle=True, validation_set=(testdata, testlabels), show_metric=True, batch_size=None, snapshot_epoch=True, run_id='fc-classifier')
#model.save("fc-classifier.tfl")
#print("Network trained and saved as fc-classifier.tfl")

model.load("fc-classifier.tfl")
result = model.evaluate(testdata, testlabels)
print("Evaluation result: %s" %result)

label = model.predict_label(testdata)
prediction = model.predict(testdata)
# print(prediction)
tp = 0
fp = 0
tn = 0
fn = 0

# determining accuracy of a comment being positive or negative
for i in range (0, len(testlabels)):
    if testlabels[i][0] == 0:
        # has to be positive
        if label[i][0] == 1:
            tp += 1
            print ("probability of comment being positive: %s" %(float(prediction[i][0])))
        else:
            fp += 1
            print ("probability of comment being positive: %s" %(float(prediction[i][1])))
```

```
144         else:
145             # has to be negative
146             if label[i][0] == 0:
147                 tn += 1
148                 print ("probability of comment being negative: %s" %(float(prediction[i][0])))
149             else:
150                 fn += 1
151                 print ("probability of comment being negative: %s" %(float(prediction[i][1])))
152 
153 
154     # for pred in prediction:
155     #     if np.argmax(pred) == 0:
156     #         print ("Negative comment")
157     #     else:
158     #         print ("Positive comment")
159 
160     # for i in range(0, len(testlabels)):
161 
162     #     if testlabels[i][0] == 1:
163     #         if prediction[i][0] > 0.5:
164     #             tp += 1
165     #         else:
166     #             fp += 1
167     #         print("probability of sentence being a positive comment: %s" %(int(prediction[i][0])))
168     #     else:
169     #         if prediction[i][1] > 0.5:
170     #             tn += 1
171     #         else:
172     #             fn += 1
173     #         print("probability of sentence being a negative comment: %s" %(int(prediction[i][1])))
174 
175     precision = float(tp / (tp + fp))
176     recall = float(tp / (tp + fn))
177     accuracy = float((tp + tn) / (tp + tn + fp + fn))
178 
179     print ("precision= %s" %precision)
180     print ("recall= %s" %recall)
181     print ("accuracy= %s" %accuracy)
```

Thank You.